Gifi's Eigenvalues

Jan de Leeuw

First created on October 27, 2019. Last update on December 26, 2019

Abstract

This is the abstract.

Note: This is a working paper which will be expanded/updated frequently. All suggestions for improvement are welcome. The directory deleeuwpdx.net/pubfolders/gifiEvals has a pdf version, the bib file, the complete Rmd file with the code chunks, and the R source code.

1 Introduction

In a gifiAnalysis (De Leeuw (2019)) we have n of **observations** on each of m **variables**. Each variable j has associated with it a (column-centered) **basis matrix** H_j and a **closed polyhedral convex cone** $\mathcal{K}_j$ in the space spanned by the columns of H_j.

For each of the variables we want to find a vector of **coefficients** z_j such that $H_j z_j \in \mathcal{K}_j$ and such that the **correlation matrix** R of the **transformed variables** $u_j = H_j z_j$ is, in some sense, **optimal**.

To actually define **optimality** in the Gifi context we associate with each gifiAnalysis a **dimension** p and a **partition** of the set $[M] = \{1, 2, \cdots, m\}$ indexing the variables into τ subsets $\mathcal{M}_1, \cdots, \mathcal{M}_\tau$. In a traditional gifiAnalysis we minimize gifiLoss, defined as

$$\sigma(X, Z, A) = \sum_{t=1}^{\tau} \text{ tr } (X - \sum_{j \in \mathcal{M}_t} H_j z_j a_j')'(X - \sum_{j \in \mathcal{M}_t} H_j z_j a_j'), \tag{1}$$

over X, over the a_j, and over the z_j satisfying the cone restrictions (De Leeuw (2019)). The algorithm is a combination of alternating least squares and majorization (a.k.a MM). The Gifi package in R implements this algorithm, for various choices of partitionings and bases.

2 Gifi Eigenvalues

In order to stay away from the trivial solution $X = 0$ and $z_j a_j' = 0$, which of course gives a zero loss, we need some form of normalization.

To discuss normalizations we use some additional notation. Define $\mathcal{H}_t$ by horizontally concatenating the H_j with $j \in \mathcal{M}_t$. Define $\mathcal{Z}_t$ as the direct sum of the z_j with $j \in \mathcal{M}_t$. Thus $\mathcal{Z}_t$ is block-diagonal, with the column-vectors z_j in the diagonal blocks. Also define $\mathcal{G}_t = \mathcal{H}_t\mathcal{Z}_t$. Finally $\mathcal{A}_t$ vertically concatenates the row vectors a_j with $j \in \mathcal{M}_t$. Thus loss function (1) becomes

$$\sigma(X, Z, A) = \sum_{t=1}^{\tau} \text{ tr } (X - \mathcal{G}_t\mathcal{A}_t)'(X - \mathcal{G}_t\mathcal{A}_t), \tag{2}$$

De Leeuw (2019) discusses three different normalizations. The first one requires $X'X = I$ but leaves A unnormalized, the second one requires $\sum_{t=1}^{\tau} \mathcal{A}_t'\mathcal{G}_t'\mathcal{G}_t\mathcal{A}_t = I$ and leaves X unnormalized. The third one requires both X and A to be normalized. All three normalizations give essentially the same solution for X, A, and Z. The three solutions only differ in the scaling of the p dimensions.

2.1 Normalize X

In the original Gifi implementation, and in the R implementation of the homals package (De Leeuw and Mair (2009)), we normalize X and leave A free. So we study this normalization first.

The minimum of loss (2) over A is

$$\min_A \sigma(X, Z, A) = p\tau - \sum_{t=1}^{\tau} \text{ tr } X'\mathcal{G}_t\,\mathcal{G}_t^+X,$$

with $\mathcal{G}_t^+$ the Moore-Penrose inverse of $\mathcal{G}_t$. Define $\mathcal{P}_t = \mathcal{G}_t\mathcal{G}_t^+$, and define $\mathcal{P}_\star$ as the average of the τ matrices $\mathcal{P}_t$. Then

$$\min_{X'X=I} \min_A \sigma(X, Z, A) = \tau(p - \max_{X'X=I} \text{tr } X'\mathcal{P}_\star X) = \tau(p - \sum_{s=1}^{p} \lambda_t(\mathcal{P}_\star)),$$

where the $\lambda_s(\mathcal{P}_\star)$ are the eigenvalues of $\mathcal{P}_\star$, ordered from large to small.

Of course we should not forget that the $\mathcal{G}_t$, and thus $\mathcal{P}_\star$ and its eigenvalues, are all functions of the z_j. Thus the problem that still must be solved, after projecting out A and X from the loss function, is to maximize the sum of the p largest eigenvalues of $\mathcal{P}_\star$ over the z_j.

2.1.1 Properties

Since $\mathcal{P}_t$ projects on the subspace $\mathcal{L}_t$ spanned by the columns of $\mathcal{G}_t$, it is idempotent and its eigenvalues are between zero and one. Thus the eigenvalues of $P_\star$ are also between zero and one. An eigenvalue is equal to one if and only if its eigenvector is in the intersection of the $\mathcal{L}_t$. In the terminology of Gifi (1990) that is if and only if it is in the **meet** of the $\mathcal{L}_t$. Also

$$0 \leq \min_{X'X=I} \min_A \sigma(X, Z, A) \leq \tau p,$$

where loss is equal to zero if and only if the $\mathcal{L}_t$ have a **p-meet**, i.e. their intersection has dimension larger than or equal to p.

2.2 Normalize A

One possible disadvantage of the original Gifi formulation is that the iterative minimization of loss has to be carried out in a space with dimension of order np, the number of observations times the number of dimensions. This can, of course, be a very large number. Partly for that reason Tijssen and De Leeuw (1989) investigated minimization of Gifi loss using normalization of A, while leaving X free.

First, observe that

$$\min_X \ \sigma(X,Z,A) = \sum_{t=1}^{\tau} \ \text{tr} \ \mathcal{A}_t'\mathcal{G}_t'\mathcal{G}_t\mathcal{A}_t - \frac{1}{\tau}\sum_{t=1}^{\tau}\sum_{s=1}^{\tau} \ \text{tr} \ \mathcal{A}_s'\mathcal{G}_s'\mathcal{G}_t\mathcal{A}_t.$$

Suppose $\mathcal{C}$ is the block matrix with blocks $\mathcal{G}_s'\mathcal{G}_t$ and $\mathcal{D}$ is the block-diagonal matrix with blocks $\mathcal{D}_t$. If Π is the $m \times m$ incidence matrix of the partition, i.e. $\pi_{j\ell} = 1$ if two variables are in the same subset and $\pi_{j\ell} = 0$ otherwise, then $\mathcal{D}$ is the elementwise or Hadamard product $\Pi \times \mathcal{C}$. With these definitions

$$\min_X \ \sigma(X,Z,A) = \text{tr} \ \mathcal{A}'\mathcal{D}\mathcal{A} - \frac{1}{\tau}\text{tr} \ \mathcal{A}'\mathcal{C}\mathcal{A},$$

and

$$\min_{\mathcal{A}'\mathcal{D}\mathcal{A}=\tau I} \min_X \ \sigma(X,Z,A) = \tau p - \sum_{s=1}^{p} \lambda_s(\mathcal{C}, \tau\mathcal{D}),$$

A gifiAnalysis find the z_j in their cones $\mathcal{K}_j$ such that the sum of the p largest eigenvalues of the generalized eigenvalue problem for the pair (R,S), with $S = \Pi\star R$, is maximized.

2.2.1 Properties

The eigenvalue problems defining a gifiAnalysis have some simple properties that are true for any partitioning of the variables.

First, note that any gifiAnalysis is **scale-free** because for every diagonal D the eigenvalues of the pair (DRD, DSD) are the same as those of (R,S). Thus, without loss of generality, we will restrict our covariance matrices to be correlation matrices.

Second, the eigenvalues of the pair (R,S) with $S = \Pi \star R$ have simple upper bounds.

Result 1: [**Upper Bound**]: If the partition Π has r sets then $0 \lesssim R \lesssim rS$.

Proof: Partition the m columns of H using Π, creating submatrices $H_1, \cdots, H_r$. Define the r linear combinations $t_q = H_q y_q$ and their average $\sqcup$. Then

$$\sum_{q=1}^{r}(t_q - \sqcup)'(t_q - \sqcup) = y'Sy - r^{-1}y'Ry \geq 0. \tag{3}$$

■

Result 2: [**Unity**] The sum of the p largest eigenvalues of R and $S = \Pi \star R$ is less than or equal to rp, with equality if and only if the r subspaces spanned by the transformed variables in the r sets have a p-dimensional intersection (or **meet**).

Proof: There is equality in (3) if and only if all t_q are equal. Thus we have $Ry = rSy$ if and only if the r subspaces have a non-trivial intersection. ■

2.2.2 Aspect Theory

The eigenvalues of the pair $(R, \Pi \star R)$ are functions of the z_j. Using the terminology used in De Leeuw (1988c) the sum of the p largest eigenvalues is an **aspect** of the correlation matrix R. A gifiAnalysis chooses the z_j to maximize that particular aspect.

A general majorization algorithm for aspects is outlined in De Leeuw (1988c), and implemented in R by Mair and De Leeuw (2010). It is proved to be convergent if the aspect is a convex functions of the correlation matrix.

If the aspect f is convex then for any two correlation matrices R and $\tilde{R}$ the majorization inequality is

$$f(R) \geq f(\tilde{R}) + \text{tr } (R - \tilde{R})\mathcal{D}f(\tilde{R}).$$

Suppose $\tilde{R}$ is the current best estimate of the optimum correlation. If we find an R such that tr $R\mathcal{D}f(\tilde{R}) >$ tr $\tilde{R}\mathcal{D}f(\tilde{R})$ then $f(R) > f(\tilde{R})$. What interests us in this paper is if the sum of the p largest eigenvalues of $(R, \Pi \star R)$ is a convex function of R, or, more precisely, for which partitions, if any, it is a convex function of R.

It is well known that the sum of the p largest eigenvalues of a real symmetric matrix of order n is a convex function of its matrix argument. This follows easily from the Ky Fan representation of that sum as $\max_X$ tr $X'RX$, where X varies over the $n \times p$ orthonormal matrices. Indeed, tr $X'RX$ is linear in R, and thus the sum of the largest eigenvalues is the maximum of a family of linear functions, which implies it is convex. The function we are optimizing is convex for the partition with exactly one variable in each set. We will study what happens for general partitions, for which the situation is less simple.

3 Derivatives

We can study convexity of Gifi's eigenvalues by computing their first and second derivatives, assuming throughout that the eigenvalues we are differentiating are simple. Even if there is no convexity these derivatives are useful for optimization, not only of our sum-of-eigenvalues function, but for other functions of the eigenvalues used in multivariate analysis, such as the ones defined by Horst (1961) or Kettenring (1971).

The correlation matrix R is a function of the coefficients z_j. We give the formulas for the derivatives of the aspect with respect to the coefficients in two steps, using the chain rule. First there is the derivative of the aspect as a function of the correlation matrix, and then there are the derivatives of the correlation matrix with respect to the coefficients.

We start with a slightly more general formulation than is needed for a gifiAnalysis. Consider the problem $Ay = \lambda By$ and $y'By = 1$ where

$$A(\theta) = A_0 + \sum_{k=1}^K \theta_k A_k,$$
$$B(\theta) = B_0 + \sum_{k=1}^K \theta_k B_k.$$

Here A_k and B_k are symmetric matrices. Thus both A and B are defined as functions of the vector with the K elements θ_k. This is more general than what is needed for Gifi, because in a gifiAnalysis A and B have very specific forms. We will specialize our results after deriving the more general ones, which are useful in other contexts (for example when deriving confidence regions for correspondence analysis solutions, see De Leeuw (2007)).

For the first and second derivatives we use the formulas in De Leeuw (2007), although the results given there also appear in hundreds of other places in the literature of functional analysis, numerical analysis, multivariate statistical analysis, optimization, and engineering. The approach in De Leeuw (2007) is similar, for example, to that of Friswell (1994), who also discusses how to generalize the formulas to higher order derivatives. Shapiro (1985) shows how to derive the required results directly from the implicit function theorem.

As an aside, there are also numerous results available, both in the mathematical and the engineering literature, for the case of multiple eigenvalues, which are generally only directionally differentiable functions of their matrix argument. For eigenvalues of symmetric matrices formulas for the second-order directional derivatives have been given by Torki (2001) and Zhang, Zhang, and Xiao (2013). We will not use these results here, and just keep them in mind for possibly later extensions.

First some definitions. Define the K vectors $(A_k - \lambda_s(\theta)B_k)y_s(\theta)$ of length n, and concatenate them vertically to the $K \times n$ matrix $H_s(\theta)$. Also define the K-element vector $g_s(\theta)$ with elements $y_s'(\theta)B_k y_s(\theta)$ and the K-element vector $f_s(\theta) = H_s(\theta)y_s(\theta)$. Finally, again following De Leeuw (2007), we define a type of generalized inverse

$$W_s(\theta) = (A(\theta) - \lambda_s(\theta)B(\theta))^- = Y(\theta)(\Lambda(\theta) - \lambda_s(\theta)I)^+ Y'(\theta), \tag{4}$$

where $Y(\theta)$ and $\Lambda(\theta)$ are complete sets of eigenvectors and eigenvalues for the pair $(A(\theta), B(\theta))$, and superscript $+$ is the Moore-Penrose inverse.

With these definitions we have nice compact matrix expressions for the derivatives.

$$\mathcal{D}\lambda_s(\theta) = H_s(\theta)y_s(\theta) = f_s(\theta), \tag{5}$$
$$\mathcal{D}y_s(\theta) = -W_s(\theta)H_s(\theta) - \frac{1}{2}y_s(\theta)g_s'(\theta), \tag{6}$$
$$\mathcal{D}^2\lambda_s(\theta) = -2H_s(\theta)W_s(\theta)H_s'(\theta) - g_s(\theta)f_s'(\theta) - f_s(\theta)g_s'(\theta). \tag{7}$$

3.1 Signature

The **signature** (sometimes called the **inertia**) of a real symmetric matrix is the triple (π, ν, δ) with the number of positive, negative, and zero eigenvalues. Sylvester's Law of Inertia says

that for any square non-singular X the signature of $X'AX$ is equal to the signature of A. In Dancis (1986) this is generalized to arbitrary X, not necessarily square or non-singular. We give a different proof of their theorem 3.1.

Result 3: [**Poincaré**] Suppose X is an $n \times m$ matrix of rank r and A is a symmetric matrix of order n. If the signature of A is (π, ν, δ) and the signature of $X'AX$ is $\tilde{\pi}, \tilde{\nu}, \tilde{\delta}$ then $\pi - (n-r) \leq \tilde{\pi} \leq \pi$ and $\nu - (n-r) \leq \tilde{\nu} \leq \nu$.

Proof: Suppose the singular value decomposition of X is

$$X = K \begin{bmatrix} \Lambda & 0 \\ 0 & 0 \end{bmatrix} L',$$

with K orthonormal of order n, L orthonormal of order m, and Λ diagonal and positive definite of order r. Then

$$X'AX = L \begin{bmatrix} \Lambda K_1' A K_1 \Lambda & 0 \\ 0 & 0 \end{bmatrix} L',$$

where K_1 is $n \times r$ with $K_1'K_1 = I$. By applying Sylvester's Law of Inertia the signature of $X'AX$ is the signature of $\Lambda K_1'AK_1\Lambda$ plus $(0, 0, m-r)$. Applying Sylvester's Law again the signature of $\Lambda K_1'AK_1\Lambda$ is the signature of $K_1'AK_1$. Now apply Poincaré's Separation Theorem (see, for example, Abadir and Magnus (2005), exercise 12.46) to deduce that the eigenvalues $\mu_1 \geq \cdots \geq \mu_r$ of $K_1'AK_1$ are related to the eigenvalues $\lambda_1 \geq \cdots \geq \lambda_n$ of A by the inequalities $\lambda_{n-r+i} \leq \mu_i \leq \lambda_i$. Since $\mu_i \leq \lambda_i$ it is clear that $\tilde{\pi} \leq \pi$. Thus $\mu_i > 0$ if $\lambda_{n-r+i} > 0$ if $n - r + i \leq \pi$ or $i \leq \pi - (n-r)$. Thus $\tilde{\pi} \geq \pi - (n-r)$. The result for ν follows by applying the same reasoning to $-A$, which of course has signature (ν, π, δ). ■

Result 4: Signature Suppose the $K \times (n+1)$ matrix $\begin{bmatrix} H_s(\theta) & g_s(\theta) \end{bmatrix}$ has rank $n+1$. Then $\mathcal{D}^2\lambda_s(\theta)$ has $n - s + 1$ positive and s negative eigenvalues.

Proof: We apply result 5 to the second derivatives $\mathcal{D}^2\lambda_s(\theta)$. Write

$$\mathcal{D}^2\lambda_s(\theta) = -\begin{bmatrix} H_s(\theta) & f_s(\theta) & g_s(\theta) \end{bmatrix} \begin{bmatrix} 2W_s(\theta) & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} H_s'(\theta) & f_s'(\theta) & g_s'(\theta) \end{bmatrix}$$

Now the signature of $W_s(\theta)$ is $(s-1, n-s, 1)$ and thus the signature of

$$\begin{bmatrix} 2W_s(\theta) & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

is $(s, n-s+1, 1)$. Taking the negative gives $(n-s+1, s, 1)$. The matrix

$$\begin{bmatrix} H_s(\theta) & f_s(\theta) & g_s(\theta) \end{bmatrix}$$

is $K \times (n+2)$, but since $f_s(\theta)$ is in the column space of $H_s(\theta)$ its rank is less than or equal to $n+1$. We assume it is actually $n+1$. Then result 5 gives us $\pi - 1 \leq \tilde{\pi} \leq \pi$ or $n - s \leq \tilde{\pi} \leq n - s + 1$. And $\nu - 1 \leq \tilde{\nu} \leq \nu$ or $s - 1 \leq \tilde{\nu} \leq s$.

Any vector z satisfying both $z'H_s(\theta) = 0$ and $z'g_s(\theta) = 0$ satisfies $\mathcal{D}^2\lambda_s(\theta)z = 0$. Since we assume that $r = n + 1$ there are $n + 1$ non-zero eigenvalues It follows that $\tilde{\pi} = n - s + 1$ and $\tilde{\nu} = s$. ■

It is abundantly clear from the results on signature that we can forget about proving for egeneral partitions the convexity of the eigenvalues and their sums.

3.2 Computation

The R function computeAB() in the appendix computes and returns the first derivative of a single eigenvalue and corresponding eigenvector with respect to θ, using formulas (5), and (6). It also computes the second derivatives of the eigenvalue, using (7). In order to make sure that everything is OK, we also use checkAB(), which uses the numDeriv package (Gilbert and Varadhan (2019)) to compute and return the same quantities using numerical differentiation.

We give the eigenvalues of the matrix of second derivatives for the partition $(1, 1, 2, 3, 3, 3)$ and for the first three eigenvalues, using both computeAB() and checkAB().

```
rab <- makeAB (c(1,1,2,3,3,3))
h <- computeAB (rab$r, rab$a, rab$b, 1)
mprint (eigen(h$hlf)$values, d = 6, w = 8, f = "+-")
```

```
##  [1] +22.434684 +17.508448 +10.774439 +7.727820  +0.580109  +0.078904
##  [7] +0.000000  +0.000000  +0.000000  +0.000000  +0.000000  -0.000000
## [13] -0.000000  -0.000000  -0.511440
```

```
h <- checkAB (rab$r, rab$a, rab$b, 1)
mprint (eigen(h$hln)$values, d = 6, w = 8, f = "+-")
```

```
##  [1] +22.434684 +17.508448 +10.774439 +7.727820  +0.580109  +0.078904
##  [7] +0.000000  +0.000000  +0.000000  -0.000000  -0.000000  -0.000000
## [13] -0.000000  -0.000000  -0.511440
```

```
h <- computeAB (rab$r, rab$a, rab$b, 2)
mprint (eigen(h$hlf)$values, d = 6, w = 8, f = "+-")
```

```
##  [1] +66.324525 +25.916113 +7.638237  +0.745604  +0.572377  +0.000000
##  [7] -0.000000  -0.000000  -0.000000  -0.000000  -0.000000  -0.000000
## [13] -0.000000  -0.606543  -18.251480
```

```
h <- checkAB (rab$r, rab$a, rab$b, 2)
mprint (eigen(h$hln)$values, d = 6, w = 8, f = "+-")
```

```
##  [1] +66.324525 +25.916113 +7.638237  +0.745604  +0.572377  +0.000000
##  [7] +0.000000  +0.000000  +0.000000  +0.000000  -0.000000  -0.000000
## [13] -0.000000  -0.606543  -18.251479
```

```
h <- computeAB (rab$r, rab$a, rab$b, 3)
mprint (eigen(h$hlf)$values, d = 6, w = 8, f = "+-")
```

```
##  [1] +59.742434 +29.991981 +10.385545 +0.043659  +0.000000  +0.000000
##  [7] +0.000000  -0.000000  -0.000000  -0.000000  -0.000000  -0.000000
## [13] -0.044127  -19.285103 -62.029367
```

```
h <- checkAB (rab$r, rab$a, rab$b, 3)
mprint (eigen(h$hln)$values, d = 6, w = 8, f = "+-")
```

```
##  [1] +59.742434 +29.991981 +10.385545 +0.043658  +0.000000  +0.000000
##  [7] +0.000000  +0.000000  +0.000000  -0.000000  -0.000000  -0.000001
## [13] -0.044128  -19.285103 -62.029367
```

Clearly the results of computeAB() and checkAB() are basically the same, except for some minor numerical differences. Moreover the signatures agree with our theoretical results. For this small example the results using numerical derivatives are quite impressive. The computing the exact (formula) results is slightly more efficient, and we expect that using formulas will have more of an advantage in efficiency and precision for larger examples.

3.3 Partials

We now specialize our results to compute partial derivatives with respect to the lower-diagonal elements of the correlation matrix $R = \{\rho_{ij}\}$, with $i > j$. Thus we have $K = \frac{1}{2}m(m-1)$, where m is the number of variables. We use double-subscripting for the K matrices A_{ij}, which are $m \times m$. They are of the form $A_{ij} = e_i e_j' + e_j e_i'$, with e_i and e_j unit vectors. The A_{ij} with $i > j$ have elements (i, j) and (j, i) equal to $+1$, while all other elements are zero. Thus

$$\begin{aligned} A(\rho) &= I + \sum_{1 \le j \le i \le m} \rho_{ij} A_{ij}, \\ B(\rho) &= I + \sum_{1 \le j \le i \le m} \rho_{ij} \pi_{ij} A_{ij}. \end{aligned}$$

There is no need to compute and store A_{ij} and B_{ij} in this case, we can simply substitute their functional form into equations (5), (6) and (7). This obviously saves both multiplications and memory.

Thus, for $i > j$, the derivatives are

$$\mathcal{D}_{(ij)}\lambda_s(\rho) = 2(1 - \lambda_s(\rho)\pi_{ij})y_{si}(\rho)y_{sj}(\rho), \tag{8}$$

and

$$\mathcal{D}_{(ij)}y_{\ell s}(\rho) = -(1 - \lambda_s(\rho)\pi_{ij})(y_{js}(\rho)w_{si\ell}(\rho) + y_{is}(\rho)w_{sj\ell}(\rho)) - \pi_{ij}y_{\ell s}(\rho)y_{is}(\rho)y_{js}(\rho), \tag{9}$$

where $W_s(\rho)$ is defined by (4).

The second derivatives are

$$\begin{aligned} &\mathcal{D}_{(ij)(kl)}\lambda_s(\rho) = \\ &\quad - 2\mathcal{D}_{(kl)}\lambda_s(\rho)\pi_{ij}y_{is}(\rho)y_{js}(\rho) + 2(1 - \lambda_s(\rho)\pi_{ij})(\mathcal{D}_{(kl)}y_{is}(\rho)y_{js}(\rho) + y_{is}(\rho)\mathcal{D}_{(kl)}y_{js}(\rho)) \end{aligned} \tag{10}$$

3.4 Computation

In the appendix we give the R function computeP() which implements formulas (??), (??) and (??). We use a small example with four variables, in two sets, to compute first and second partials with both computeAB() and computeP().

```
p <- c(1, 1, 2, 2)
rab <- makeAB(p)
hab <- computeAB (rab$r, rab$a, rab$b, 1)
hpa <- computeP (rab$r, p, 1)
```

We now compare the two results to check if our formulas and our computations are correct. The maximum absolute difference between the two results for the gradient of the eigenvalue is $1.0408340856 \times 10^{-17}$. For the gradient of the eigenvector it is $2.4980018054 \times 10^{-15}$, and for the hessian it is $4.0523140399 \times 10^{-15}$. Again, basically the same, so things seem to be OK.

3.5 Special Cases

3.5.1 Component Analysis

If $\pi_{ij} = 0$ for all $i \neq j$ (as in the case of nonlinear principal component analysis)

$$\mathcal{D}^2\lambda_s(\theta) = -2H_s'(\theta)(A(\theta) - \lambda_s(\theta)I)^+H_s(\theta)$$

It follows, assuming that $H_s(\theta)$ is of rank n, that the signature of $\mathcal{D}^2\lambda_s(\theta)$ is $(n - s, s - 1, \frac{1}{2}(n-1)(n-2))$. Thus the largest eigenvalue is a convex function and the smallest is a concave function of the matrix argument.

We verify the signature using computeP() on a Spearman correlation matrix of order six, for the first three eigenvalues.

```
p <- 1:6
z <- (1:6)/10
r <- outer (z, z)[outer(1:6, 1:6, "<")]
h1 <- computeP (r, p, 1)
mprint (eigen(h1$hlf)$values, d = 6, w = 8, f = "+-")
```

```
##  [1] +2.787716 +2.321184 +1.756067 +1.252285 +0.880824 +0.000000 +0.000000
##  [8] +0.000000 +0.000000 +0.000000 -0.000000 -0.000000 -0.000000 -0.000000
## [15] -0.000000
```

```
h2 <- computeP (r, p, 2)
mprint (eigen(h2$hlf)$values, d = 6, w = 8, f = "+-")
```

```
##  [1] +51.743207 +20.223939 +10.841689 +6.722973  +0.000000  +0.000000
##  [7] +0.000000  +0.000000  +0.000000  -0.000000  -0.000000  -0.000000
## [13] -0.000000  -0.000000  -2.782146
```

```
h3 <- computeP (r, p, 3)
mprint (eigen(h3$hlf)$values, d = 6, w = 8, f = "+-")
```

```
##  [1] +27.249588 +12.931445 +7.521237  +0.000000  +0.000000  +0.000000
##  [7] +0.000000  +0.000000  +0.000000  -0.000000  -0.000000  -0.000000
## [13] -0.000000  -2.247776  -51.689503
```

We can also check that the sum of the first two and the sum of the first three eigenvalues is convex.

```
mprint (eigen(h1$hlf+h2$hlf)$values, d = 6, w = 8, f = "+-")
```

```
##  [1] +51.763142 +20.233969 +10.848115 +6.727161  +2.305156  +1.747129
##  [7] +1.246273  +0.876793  +0.000000  +0.000000  +0.000000  -0.000000
## [13] -0.000000  -0.000000  -0.000000
```

```
mprint (eigen(h1$hlf+h2$hlf+h3$hlf)$values, d = 6, w = 8, f = "+-")
```

```
##  [1] +27.518920 +20.165318 +12.995857 +10.841753 +7.548198  +6.725746
##  [7] +1.648177  +1.211152  +0.857608  +0.000000  +0.000000  +0.000000
## [13] -0.000000  -0.000000  -0.000000
```

3.5.2 Regression

Suppose there are two sets and the first set contains only a single variable (as in the case of nonlinear multiple regression). We can partition the correlation matrix as

$$\begin{bmatrix} 1 & r' \\ r & R \end{bmatrix}.$$

The largest eigenvalue is $1+\sqrt{r'Rr}$, the smallest $1-\sqrt{r'R^{-1}r}$, and there are $n-2$ eigenvalues equal to 1. Now $1+\sqrt{r'Rr}$ is not a convex function of the correlation matrix, but $r'R^{-1}r$ is convex (De Leeuw (1988c)). It follows that $(1-\lambda_1(\theta))^2$ is convex. Obviously maximizing the largest eigenvalue is the same as maximizing the aspect $r'R^{-1}r$.

Our usual result on signatures applies in the regression case as well.

```
p <- 1:6
z <- (1:6)/10
r <- outer (z, z) + diag(1 - z ^ 2)
mprint(r)
```

```
##      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
## [1,] +1.000000 +0.020000 +0.030000 +0.040000 +0.050000 +0.060000
## [2,] +0.020000 +1.000000 +0.060000 +0.080000 +0.100000 +0.120000
## [3,] +0.030000 +0.060000 +1.000000 +0.120000 +0.150000 +0.180000
## [4,] +0.040000 +0.080000 +0.120000 +1.000000 +0.200000 +0.240000
## [5,] +0.050000 +0.100000 +0.150000 +0.200000 +1.000000 +0.300000
## [6,] +0.060000 +0.120000 +0.180000 +0.240000 +0.300000 +1.000000
```

```
lr <- eigen (r)$values[1]
r <- r[outer(1:6, 1:6, "<")]
p <- c(1,2,2,2,2,2)
hp <- computeP (r, p, 1)
mprint (eigen(hp$hlf)$values, d = 6, w = 8, f = "+-")
```

```
##  [1] +19.356464 +16.685695 +15.117584 +14.179460 +0.290875  +0.000000
##  [7] +0.000000  +0.000000  +0.000000  +0.000000  +0.000000  +0.000000
## [13] -0.000000  -0.000000  -0.262528
```

```
hq <- computeP (r, p, 2)
mprint(hq$glf)
```

```
##  [1] -0.000000 -0.000000 -0.000000 +0.000000 +0.000000 +0.000000 -0.000000
##  [8] -0.000000 -0.000000 +0.000000 +0.000000 +0.000000 +0.000000 -0.000000
## [15] +0.000000
```

We expect convexity for $\rho(\theta) = (1 - \lambda_1(\theta))^2$. Now

$$\mathcal{D}^2\rho(\theta) = -2(1 - \lambda_1(\theta))\mathcal{D}^2\lambda_1(\theta) + 2\mathcal{D}\lambda_1(\theta)(\mathcal{D}\lambda_1(\theta))',$$

and we can thus use the results of computeAB() to verify convexity in the regression case.

```
hlmp <- -2 * ( 1 - lr) * hp$hlf + 2 * outer (hp$glf, hp$glf)
mprint (eigen(hlmp)$values, d = 6, w = 8, f = "+-")
```

```
##  [1] +26.069560 +22.469558 +20.355243 +19.090233 +1.353669  +0.000000
##  [7] +0.000000  -0.000000  -0.000000  -0.000000  -0.000000  -0.000000
## [13] -0.000000  -0.000000  -0.090875
```

```
mab <- makeAB(p)
hab <- computeAB(r, mab$a, mab$b, 1)
print(max(abs(hp$glf - hab$glf)))
```

```
## [1] 6.938893904e-18
```

```
print(max(abs(hp$gyf - hab$gyf)))
```

```
## [1] 3.330669074e-16
```

```
print(max(abs(hp$hlf - hab$hlf)))
```

```
## [1] 4.996003611e-16
```

```
hlmab <- -2 * ( 1 - lr) * hab$hlf + 2 * outer (hab$glf, hab$glf)
print(max(abs(hlmp - hlmab)))
```

```
## [1] 7.21644966e-16
```

```
mprint (eigen(hlmab)$values, d = 6, w = 8, f = "+-")
```

```
##  [1] +26.069560 +22.469558 +20.355243 +19.090233 +1.353669  +0.000000
##  [7] +0.000000  +0.000000  +0.000000  +0.000000  -0.000000  -0.000000
## [13] -0.000000  -0.000000  -0.090875
```

4 GifiA Derivatives

In a gifiAnalysis

$$\rho_{ij}(z) = \frac{z_i' C_{ij} z_j}{\sqrt{z_i' D_i z_i}\sqrt{z_j' D_j z_j}},$$

where the $C_{ij} = H_i' H_j$ are submatrices of the Burt matrix (remember the H_i are column-centered). By the chain rule

$$\mathcal{D}_k \lambda_s(z) = \sum_{1\leq j<i\leq n} \mathcal{D}_{(ij)} \lambda_s(\rho(z)) \mathcal{D}_k \rho_{ij}(z). \tag{11}$$

The function computeZ() combines (11) with

$$\mathcal{D}_i \rho_{ij}(z) = \frac{1}{\sqrt{z_i' D_i z_i}}(I - D_i \tilde{z}_i \tilde{z}_i')C_{ij}\tilde{z}_j, \tag{12}$$

$$\mathcal{D}_j \rho_{ij}(z) = \frac{1}{\sqrt{z_j' D_j z_j}}(I - D_j \tilde{z}_j \tilde{z}_j')C_{ji}\tilde{z}_i, . \tag{13}$$

This is not the most efficient way of computing the partials, but it gets the job done. In the same way, by the chain rule,

$$\mathcal{D}_k y_s(z) = \sum_{1\leq j<i\leq n} \mathcal{D}_{(ij)} y_s(\rho(z)) \mathcal{D}_k \rho_{ij}(z). \tag{14}$$

For the second partials we have, again using the chain rule,

$$\mathcal{D}_{k\ell} \lambda_s(z) = \sum_{j<i}\sum_{q<p} \mathcal{D}_{(ij)(pq)} \lambda_s(\rho(z)) \mathcal{D}_k \rho_{ij}(z) \mathcal{D}_\ell \rho_{pq}(z) + \sum_{j<i} \mathcal{D}_{(ij)} \lambda_s(\rho(z)) \mathcal{D}_{k\ell} \rho_{ij}(z). \tag{15}$$

Once again, the computeZ() function does not aim to be efficient. It directly implements the chain rule $(15), using the partials from computeP(), and in addition

$$\mathcal{D}_{ij} \rho_{ij}(z) = \frac{1}{\sqrt{z_i' D_i z_i}\sqrt{z_j' D_j z_j}}(I - D_i \tilde{z}_i \tilde{z}_i')C_{ij}(I - \tilde{z}_j \tilde{z}_j' D_j), \tag{16}$$

$$\mathcal{D}_{ii} \rho_{ij}(z) = \frac{1}{z_i' D_i z_i}\left[3\rho_{ij} D_i \tilde{z}_i \tilde{z}_i' D_i - (D_i \tilde{z}_i \tilde{z}_j' C_{ji} + C_{ij} \tilde{z}_j \tilde{z}_i' D_i) - \rho_{ij} D_i\right], \tag{17}$$

$$\mathcal{D}_{jj} \rho_{ij}(z) = \frac{1}{z_j' D_j z_j}\left[3\rho_{ij} D_j \tilde{z}_j \tilde{z}_j' D_j - (D_j \tilde{z}_j \tilde{z}_i' C_{ij} + C_{ji} \tilde{z}_i \tilde{z}_j' D_j) - \rho_{ij} D_j\right]. \tag{18}$$

4.1 Bilinearizability

It is interesting that if there are z_j with $z_j'D_jz_j = 1$ that satisfy the simultaneous bilinear regression conditions $C_{kj}z_j = r_{kj}D_kz_k$ (see De Leeuw (1988b) and De Leeuw (1988a)) then by (9) and (10) $\mathcal{D}_k\rho_{ij}(z) = 0$ and thus $\mathcal{D}_k\lambda_s(z) = 0$ and $\mathcal{D}_k y_s(z)$, no matter what the partition is. It follows that a system of bilinearizing z_j corresponds with a stationary solution for any aspect that is a function of the correlation coefficients.

Not necessarily a local maximum of minimum, of course. In fact from (16), (17) and (18) we see that

$$\begin{aligned}\mathcal{D}_{ij}\rho_{ij}(z) &= C_{ij} - \rho_{ij}(z)D_iz_iz_j'D_j,\\ \mathcal{D}_{ii}\rho_{ij}(z) &= \rho_{ij}(z)(D_i - D_iz_iz_i'D_i),\\ \mathcal{D}_{jj}\rho_{ij}(z) &= \rho_{ij}(z)(D_j - D_jz_jz_j'D_j).\end{aligned}$$

Thus, for simultaneously bilinearizing scores z_j,

$$\mathcal{D}_{k\ell}\lambda_s(z) = 2\sum_{j<i}(1 - \lambda_s(\rho(z))\pi_{ij})y_{si}(\rho(z))y_{sj}(\rho(z))\mathcal{D}_{k\ell}\rho_{ij}(z). \tag{19}$$

4.2 Computation

```
icat <- c (2, 2, 4, 4)
p <- c (1, 1, 2, 2)
hz <- makeHZ (100, 4, icat)
cf <- computeZ (hz$z, hz$h, hz$r, p, 1)
cn <- checkZ (hz$z, hz$h, p, 1)
print(max(abs(unlist(cf$glf) - cn$gln)))
```

```
## [1] 1.473018209e-10
```

```
print(max(abs(matList(cf$gyf) - cn$gyn)))
```

```
## [1] 0.8471463466
```

```
h1 <- ifelse (outer(sample (1:4, 100, replace = TRUE), 1:4, "=="), 1, 0)
h2 <- ifelse (outer(sample (1:6, 100, replace = TRUE), 1:6, "=="), 1, 0)
h1 <- apply (h1, 2, function (x) x - mean (x))
h2 <- apply (h2, 2, function (x) x - mean (x))
h1 <- svd (h1)$u[, -4, drop = FALSE]
h2 <- svd (h2)$u[, -6, drop = FALSE]
c12 <- crossprod (h1, h2)
s <- svd (c12)
z1 <- s$u[,1]
z2 <- s$v[,1]
h <- list (h1, h2)
z <- list (z1, z2)
```

```
r <- z1 %*% c12 %*% z2
mprint(r)
```

```
##       [,1]
## [1,] +0.238971
```

```
p <- c(1,2)
zcompu <- computeZ(z, h, r, p, 1)
hess <- matrix(0, 8, 8)
hess[1:3,1:3]<-zcompu$hlf[[1]]
hess[4:8,1:3]<-zcompu$hlf[[2]]
hess[4:8,4:8]<-zcompu$hlf[[3]]
hess[1:3,4:8]<-t(zcompu$hlf[[2]])
mprint(eigen(hess)$values)
```

```
## [1] -0.000000 -0.000000 -0.043703 -0.139061 -0.238971 -0.238971 -0.338882
## [8] -0.434239
```

```
zcompu <- computeZ(z, h, r, p, 2)
hess[1:3,1:3]<-zcompu$hlf[[1]]
hess[4:8,1:3]<-zcompu$hlf[[2]]
hess[4:8,4:8]<-zcompu$hlf[[3]]
hess[1:3,4:8]<-t(zcompu$hlf[[2]])
mprint(eigen(hess)$values)
```

```
## [1] +0.434239 +0.338882 +0.238971 +0.238971 +0.139061 +0.043703 +0.000000
## [8] +0.000000
```

5 Appendix: Duality Gap

The result in this appendix may not be of much practical use, but it's a nice example of the duality gap, so I'll include it anyway. The primal problem is

$$\max_{X'BX=I} \text{ tr } X'AX = \max_X \min_{M\in\mathcal{S}^n} \text{tr } X'AX - \text{tr } M(X'BX - I).$$

We now study the Lagrange dual problem

$$\min_{M\in\mathcal{S}^n} \max_X \text{tr } X'AX - \text{tr } M(X'BX - I).$$

Using Kronecker products this becomes

$$\max_x \text{tr } x'[(I\otimes A) - (M\otimes B)]x + \text{tr } M = \begin{cases} \text{tr } M & \text{if } M\otimes B \gtrsim I\otimes A, \\ +\infty & \text{otherwise.} \end{cases}$$

Now

$$\min_{M\in\mathcal{S}^n}\{\text{tr } M \mid M\otimes B \gtrsim I\otimes A\} = p\lambda_p(B^{-1}A),$$

and thus

$$\min_{M\in\mathcal{S}^n}\max_X \text{tr } X'AX - \text{tr } M(X'BX - I) \leq \max_X \min_{M\in\mathcal{S}^n} \text{tr } X'AX - \text{tr } M(X'BX - I),$$

with equality if and only if the p largest eigenvalues of $B^{-1}A$ are all equal.

6 Appendix: MaxMax Duality

There is another result which I am just reporting here because I have it sitting around. First

$$\text{tr } X'AX = \max_Y 2 \text{ tr } Y'AX - \text{tr } Y'AY.$$

Thus

$$\max_{X'BX=I} \text{tr } X'AX = \max_{X'BX=I}\max_Y 2 \text{ tr } Y'AX - \text{tr } Y'AY = \max_Y \max_{X'BX=I} 2 \text{ tr } Y'AX - \text{tr } Y'AY$$

Now

$$\max_{X'BX=I} 2 \text{ tr } Y'AX - \text{tr } Y'AY = 2 \text{ tr } (Y'AB^{-1}AY)^{\frac{1}{2}} - \text{tr } Y'AY,$$

and thus

$$\max_{X'BX=I} \text{tr } X'AX = \max_Y 2 \text{ tr } (Y'AB^{-1}AY)^{\frac{1}{2}} - \text{tr } Y'AY.$$

Note that tr $(Y'AB^{-1}AY)^{\frac{1}{2}}$ is the sum of the singular values, i.e. the trace norm, of $Y'AB^{-\frac{1}{2}}$. Thus we have replaced a constrained problem with an, admittedly more complex, unconstrained problem.

7 Appendix: Code

7.1 auxiliary.R

```
library(numDeriv)

repList <- function(x, n) {
  z <- list()
  for (i in 1:n)
    z <- c(z, list(x))
  return(z)
}

sumList <- function (x, a) {
  m <- length (x)
  nr <- nrow(a[[1]])
  nc <- ncol(a[[1]])
  sa <- matrix (0, nr, nc)
```

```
  for (j in 1:m) {
    sa <- sa + x[j] * a[[j]]
  }
  return (sa)
}

matList <- function (x) {
  m <- length (x)
  n <- nrow (x[[1]])
  a <- matrix(0,n,0)
  for (j in 1:m) {
    a <- cbind(a, x[[j]])
  }
  return (a)
}

psplit <- function (z, s) {
  g <- 1 + rowSums (outer (1:length(z), cumsum(s), ">"))
  h <- split (z, g)
  names (h) <- NULL
  return (h)
}

geig <- function (a, b) {
  s <- solve (chol (b))
  h <- crossprod (s, a %*% s)
  e <- eigen (h)
  y <- s %*% e$vectors
  return (list (values = e$values, vectors = y))
}

mprint <- function (x, d = 6, w = 8, f = "+-") {
  print (noquote (formatC (
    x, di = d, wi = w, fo = "f", flag = f
  )))
}
```

7.2 computeAB.R

```
makeAB <- function (p) {
  m <- length (p)
  rmat <- cor (matrix (rnorm (1000 * m), 1000, m))
  r <- rmat[outer(1:m, 1:m, "<")]
```

```
  a <- repList(matrix(0, m, m), m * (m - 1) / 2)
  b <- repList(matrix(0, m, m), m * (m - 1) / 2)
  k <- 1
  for (i in 2:m) {
    for (j in 1:(i - 1)) {
      a[[k]][i, j] <- 1
      a[[k]][j, i] <- 1
      if (p[i] == p[j]) {
        b[[k]][i, j] <- 1
        b[[k]][j, i] <- 1
      }
      k <- k + 1
    }
  }
  return (list (r = r, a = a, b = b))
}

computeAB <- function (x, a, b, s) {
  m <- length (x)
  n <- nrow (a[[1]])
  ax <- sumList (x, a) + diag (n)
  bx <- sumList (x, b) + diag (n)
  e <- geig (ax, bx)
  l <- e$values
  y <- e$vectors
  ys <- y[, s, drop = TRUE]
  v <- ifelse (s == 1:n, 0, 1 / (l - l[s]))
  w <- tcrossprod (y %*% diag (v), y)
  hm1 <- sapply(a, function (x)
    x %*% ys)
  hm2 <- sapply(b, function (x)
    x %*% ys)
  hmat <- hm1 - l[s] * hm2
  gvec <- sapply (b, function (x)
    drop (ys %*% x %*% ys))
  gyf <- -w %*% hmat - outer (ys, gvec) / 2
  glf <- drop (ys %*% hmat)
  hlf <-
    -2 * crossprod (hmat, w %*% hmat) - outer (glf, gvec) - outer (gvec, glf)
  return (list (
    glf = glf,
    gyf = gyf,
    hlf = hlf
  ))
```

```
}

checkAB <- function (x, a, b, s) {
  fl <- function (x, a, b, s) {
    n <- nrow(a[[1]])
    ax <- sumList (x, a) + diag (n)
    bx <- sumList (x, b) + diag (n)
    e <- geig (ax, bx)
    return (e$values[s])
  }
  fy <- function (x, a, b, s) {
    n <- nrow(a[[1]])
    ax <- sumList (x, a) + diag (n)
    bx <- sumList (x, b) + diag (n)
    e <- geig (ax, bx)
    k <- which.max (abs (e$vectors[, s]))
    return (sign (e$vectors[k, s]) * e$vectors[, s])
  }
  gln <- grad (
    fl,
    x,
    side = NULL,
    a = a,
    b = b,
    s = s
  )
  gyn <- jacobian (
    fy,
    x,
    side = NULL,
    a = a,
    b = b,
    s = s
  )
  hln <- hessian (fl, x, a = a, b = b, s = s)
  return (list (gln = gln,
                gyn = gyn,
                hln = hln))
}
```

7.3 computeP.R

```
computeP <- function (r, p, s) {
  m <- length (p)
  mm <- m * (m - 1) / 2
  a <- matrix (0, m, m)
  a[outer(1:m, 1:m, "<")] <- r
  a <- a + t(a)
  diag(a) <- 1
  pp <- ifelse(outer(p, p, "=="), 1, 0)
  b <- pp * a
  e <- geig (a, b)
  l <- e$values
  y <- e$vectors
  ys <- y[, s, drop = TRUE]
  ls <- l[s]
  v <- ifelse (s == 1:m, 0, 1 / (l - ls))
  w <- tcrossprod (y %*% diag (v), y)
  gg <- rep (0, mm)
  glf <- rep (0, mm)
  gyf <- matrix (0, m, mm)
  hlf <- matrix (0, mm, mm)
  k <- 1
  for (i in 2:m) {
    for (j in 1:(i - 1)) {
      glf[k] <- 2 * (1 - pp[i, j] * ls) * ys[i] * ys [j]
      aux <-
        -(1 -  pp[i, j] * ls) * ((ys[j] * w[, i]) + (ys[i] * w[, j]))
      gyf[, k] <- aux - pp[i, j] * ys[i] * ys[j] * ys
      l <- 1
      for (u in 2:m) {
        for (v in 1:(u - 1)) {

        }
      }
      k <- k + 1
    }
  }
  k <- 1
  for (i in 2:m) {
    for (j in 1:(i - 1)) {
      fac1 <- -2 * pp[i, j] * ys[i] * ys[j]
      fac2 <-  2 * (1 - pp[i, j] * ls)
      l <- 1
      for (u in 2:m) {
        for (v in 1:(u - 1)) {
```

```
          fac3 <- (ys[j] * gyf[i, l]) + (ys[i] * gyf [j, l])
          hlf[k, l] <- fac1 * glf[l] + fac2 * fac3
          l <- l + 1
        }
      }
      k <- k + 1
    }
  }
  return (list (glf = glf,
                gyf = gyf,
                hlf = hlf))
}

checkP <- function (r, p, s) {
  fl <- function (r, p, s) {
    m <- length (p)
    mm <- m * (m - 1) / 2
    a <- matrix (0, m, m)
    a[outer(1:m, 1:m, "<")] <- r
    a <- a + t(a)
    diag(a) <- 1
    pp <- ifelse(outer(p, p, "=="), 1, 0)
    b <- pp * a
    e <- geig (a, b)
    return (e$values[s])
   }
  fy <- function (r, p, s) {
    m <- length (p)
    mm <- m * (m - 1) / 2
    a <- matrix (0, m, m)
    a[outer(1:m, 1:m, "<")] <- r
    a <- a + t(a)
    diag(a) <- 1
    pp <- ifelse(outer(p, p, "=="), 1, 0)
    b <- pp * a
    e <- geig (a, b)
    k <- which.max (abs (e$vectors[, s]))
    return (sign (e$vectors[k, s]) * e$vectors[, s])
  }
  gln <- grad (
    fl,
    x,
    side = NULL,
    a = a,
```

```
    b = b,
    s = s
  )
  gyn <- jacobian (
    fy,
    x,
    side = NULL,
    a = a,
    b = b,
    s = s
  )
  hln <- hessian (fl, x, a = a, b = b, s = s)
  return (list (gln = gln,
                gyn = gyn,
                hln = hln))
}
```

7.4 computeZ.R

```
makeHZ <- function (n, m, k) {
  h <- as.list (1:m)
  z <- as.list (1:m)
  if (length(k) == 1) {
    k <- rep (k, m)
  }
  hzz <- matrix (0, n, m)
  for (j in 1:m) {
    hh <-
      ifelse (outer (sample (1:k[j], n, replace = TRUE), 1:k[j], "=="), 1, 0)
    mh <- apply (hh, 2, mean)
    hh <- hh - outer (rep(1, n), mh)
    hh <- svd (hh)$u[, -k[j], drop = FALSE]
    zz <- rnorm (k[j] - 1)
    hz <- drop (hh %*% zz)
    zz <- zz / sqrt (sum (hz ^ 2))
    hzz[, j] <- hz
    h[[j]] <- hh
    z[[j]] <- zz
  }
  r <- cor (hzz)[outer(1:m, 1:m, "<")]
  return (list(h = h, z = z, r = r))
}

ff <- function (z, h, p) {
```

```
  m <- length (h)
  n <- nrow (h[[1]])
  hz <- matrix (0, n, m)
  for (j in 1:m) {
    hz[, j] <- h[[j]] %*% z[[j]] / sqrt (sum (z[[j]] ^ 2))
  }
  ax <- crossprod (hz)
  pp <- ifelse (outer (p, p, "=="), 1, 0)
  bx <- pp * ax
  e <- geig (ax, bx)
  return (e)
}

computeZ <- function (z, h, r, p, s) {
  m <- length (z)
  cp <- computeP (r, p, s)
  hlf <- as.list (1:(m * (m + 1) / 2))
  glf <- as.list (1:m)
  gyf <- as.list (1:m)
  kl <- 1
  for (k in 1:m) {
    drk <- drmat (z, h, k)
    glf[[k]] <- drop (cp$glf %*% drk)
    gyf[[k]] <- cp$gyf %*% drk
    for (l in 1:k) {
      drl <- drmat (z, h, l)
      hlf[[kl]] <- crossprod (drk, cp$hlf %*% drl)
      dkl <- ddmat (z, h, k, l)
      hlf[[kl]] <- hlf[[kl]] + sumList (cp$glf, dkl)
      kl <- kl + 1
    }
  }
  return (list (glf = glf, gyf = gyf, hlf = hlf))
}

drmat <- function (z, h, k) {
  m <- length (z)
  mm <- m * (m - 1) / 2
  dr <- matrix (0, mm, length (z[[k]]))
  l <- 1
  for (i in 2:m) {
    for (j in 1:(i - 1)) {
      if (i == k) {
        dr[l,] <- dirij (z, h, i, j)
```

```
      }
      if (j == k) {
        dr[l,] <- djrij (z, h, i, j)
      }
      l <- l + 1
    }
  }
  return (dr)
}

ddmat <- function (z, h, k, l) {
  m <- length (z)
  mm <- m * (m - 1) / 2
  dd <- repList(matrix (0, length(z[[k]]), length(z[[l]])), mm)
  kl <- 1
  for (i in 2:m) {
    for (j in 1:(i - 1)) {
      if ((i == k) && (j == l)) {
        dd[[kl]] <- dijrij (z, h, i, j)
      }
      if ((i == k) && (i == l)) {
        dd[[kl]] <- diirij (z, h, i, j)
      }
      if ((j == k) && (j == l)) {
        dd[[kl]] <- djjrij (z, h, i, j)
      }
      kl <- kl + 1
    }
  }
  return (dd)
}


dirij <- function (z, h, i, j) {
  cij <- crossprod (h[[i]], h[[j]])
  cii <- crossprod (h[[i]])
  czj <- drop (cij %*% z[[j]])
  return (czj - sum (czj * z[[i]]) * drop (cii %*% z[[i]]))
}

djrij <- function (z, h, i, j) {
  cji <- crossprod (h[[j]], h[[i]])
  cjj <- crossprod (h[[j]])
  czi <- drop (cji %*% z[[i]])
```

```
  return (czi - sum (czi * z[[j]]) * drop (cjj %*% z[[j]]))
}

diirij <- function (z, h, i, j) {
  cij <- crossprod (h[[i]], h[[j]])
  cii <- crossprod (h[[i]])
  czj <- drop (cij %*% z[[j]])
  czi <- drop (cii %*% z[[i]])
  rij <- sum (z[[i]] * czj)
  dii <- 3 * rij * outer (czi, czi)
  dii <- dii - outer (czj, czi)
  dii <- dii - outer (czi, czj)
  dii <- dii - rij * cii
  return (dii)
}

djjrij <- function (z, h, i, j) {
  cji <- crossprod (h[[j]], h[[i]])
  cjj <- crossprod (h[[j]])
  czi <- drop (cji %*% z[[i]])
  czj <- drop (cjj %*% z[[j]])
  rij <- sum (z[[j]] * czi)
  djj <- 3 * rij * outer (czj, czj)
  djj <- djj - outer (czi, czj)
  djj <- djj - outer (czj, czi)
  djj <- djj - rij * cjj
  return (djj)
}

dijrij <- function (z, h, i, j) {
  cij <- crossprod (h[[i]], h[[j]])
  cii <- crossprod (h[[i]])
  cjj <- crossprod (h[[j]])
  szi <- drop (t(cij) %*% z[[i]])
  szj <- drop (cij %*% z[[j]])
  czj <- drop (cjj %*% z[[j]])
  czi <- drop (cii %*% z[[i]])
  rij <- sum (z[[i]] * szj)
  dij <- cij + rij * outer(czi, czj)
  dij <- dij - outer (czi, szi)
  dij <- dij - outer (szj, czj)
  return (dij)
}
```

```
checkZ <- function (z, h, p, s) {
  fl <- function (z, h, p, s) {
    e <- ff(psplit (z, sapply (h, ncol)), h, p)
    return (e$values[s])
  }
  fy <- function (z, h, p, s) {
    e <- ff(psplit (z, sapply (h, ncol)), h, p)
    k <- which.max (abs (e$vectors[, s]))
    return (sign (e$vectors[k, s]) * e$vectors[, s])
  }
  gln <- grad (
    fl,
    unlist (z),
    side = NULL,
    h = h,
    p = p,
    s = s
  )
  gyn <- jacobian (
    fy,
    unlist(z),
    side = NULL,
    h = h,
    p = p,
    s = s
  )
  hln <- hessian (fl,
                  unlist(z),
                  h = h,
                  p = p,
                  s = s)
  return (list (gln = gln,
                gyn = gyn,
                hln = hln))
}
```

References

Abadir, K. M., and J. R. Magnus. 2005. *Matrix Algebra*. Cambridge University Press.

Dancis, J. 1986. "A Quantitative Formulation of Sylvester's Law of Inertia." *Linear Algebra and Its Applications* 80: 141–58.

De Leeuw, J. 1988a. "Model Selection in Multinomial Experiments." In *On Model Uncertainty*

and Its Statistical Implications, edited by T. K. Dijkstra. Berlin: Springer. http://deleeuwpdx.net/janspubs/1988/chapters/deleeuw_C_88a.pdf.

———. 1988b. "Multivariate Analysis with Linearizable Regressions." *Psychometrika* 53: 437–54. http://deleeuwpdx.net/janspubs/1988/articles/deleeuw_A_88a.pdf.

———. 1988c. "Multivariate Analysis with Optimal Scaling." In *Proceedings of the International Conference on Advances in Multivariate Statistical Analysis*, edited by S. Das Gupta and J. K. Ghosh, 127–60. Calcutta, India: Indian Statistical Institute. http://deleeuwpdx.net/janspubs/1988/chapters/deleeuw_C_88b.pdf.

———. 2007. "Derivatives of Generalized Eigen Systems with Applications." Preprint Series 528. Los Angeles, CA: UCLA Department of Statistics. http://deleeuwpdx.net/janspubs/2007/reports/deleeuw_R_07c.pdf.

———. 2019. "Gifi Update Notes." 2019. http://deleeuwpdx.net/pubfolders/gifi/Gifi_New/gifi.pdf.

De Leeuw, J., and P. Mair. 2009. "Homogeneity Analysis in R: The Package Homals." *Journal of Statistical Software* 31 (4): 1–21. http://deleeuwpdx.net/janspubs/2009/articles/deleeuw_mair_A_09a.pdf.

Friswell, M. I. 1994. "Calculation of Second and Higher Order Eigenvector Derivatives." *Journal of Guidance, Control, and Dynamics* 18 (4): 919–21.

Gifi, A. 1990. *Nonlinear Multivariate Analysis*. New York, N.Y.: Wiley.

Gilbert, P., and R. Varadhan. 2019. *numDeriv: Accurate Numerical Derivatives*. https://CRAN.R-project.org/package=numDeriv.

Horst, P. 1961. "Relations Among m Sets of Measures." *Psychometrika* 26: 129–49.

Kettenring, J. R. 1971. "Canonical Analysis of Several Sets of Variables." *Biometrika* 58: 433–51.

Mair, P., and J. De Leeuw. 2010. "A General Framework for Multivariate Analysis with Optimal Scaling: The R Package Aspect." *Journal of Statistical Software* 32 (9): 1–23. http://deleeuwpdx.net/janspubs/2010/articles/mair_deleeuw_A_10.pdf.

Shapiro, A. 1985. "Second-Order Derivatives of Extremal-Value Functions and Optimality Conditions for Semi-Infinite Programs." *Mathematics of Operations Research* 10 (2): 207–19.

Tijssen, R. J. W., and J. De Leeuw. 1989. "Multi-Set Nonlinear Canonical Analysis via the Burt Matrix." In *Multiway Data Analysis*, edited by R. Coppi and S. Bolasko. Amsterdam; New York: North Holland Publishing Company. http://deleeuwpdx.net/janspubs/1989/chapters/tijssen_deleeuw_C_89.pdf.

Torki, M. 2001. "Second-order Directional Derivatives of All Eigenvalues of a Symmetric Matrix." *Nonlinear Analysis* 46: 1133–50.

Zhang, L., N. Zhang, and X. Xiao. 2013. "On the Second-order Directional Derivatives of Singular Values of Matrices and Symmetric Matrix-valued Functions." *Set-Valued and*

Variational Analysis 21: 557–86.